Using HPC Batch Systems Efficiently

HPC batch system is like **traffic cop**

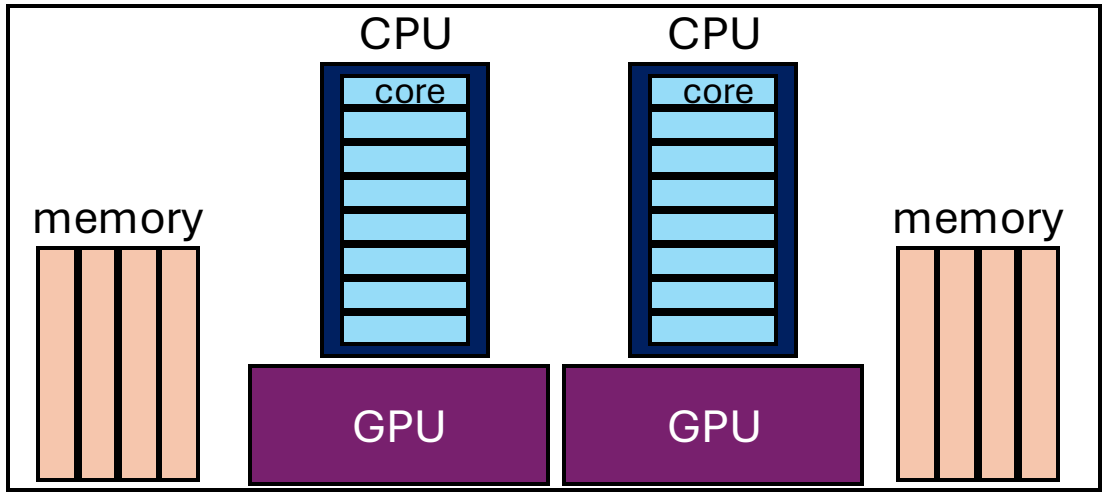
Batch system has rules that determine which jobs run when

Rules can include:

- **Fairshare** (diminishing priority based on recent use)
- **Quality of service** (select users get more access or priority)
- **Size** (large parallel jobs get higher priority)
- **Age** (jobs waiting longer get higher priority)

Very important to request correct resources!

- # nodes
- # cores (per node)
- Memory
- # GPUs
- Licenses / filesystem



Asking for more resources than you need slows everyone down!

Selecting HPC Resources

Questions to ask before submitting a job:

- Is the code serial or parallel?
- If parallel, how?
- How much memory do I need, is the default enough?
- Do I need a GPU?
- Are there any other resources I need to request?

Parallelism:

- Serial (one core)
- Shared-memory parallel (one node, multiple cores)
- Distributed-memory parallel (multiple nodes and cores)
- Hybrid

Memory use:

Batch systems often grant some amount of memory per core allocated. e.g. for a 16-core node, each core allocated may get ~1/16 of the total memory per core allocation.

GPU use:

Be aware that GPUs are usually in demand so try to use nodes with no GPUs if you don't need it.

Additional resources:

Keep in mind that some HPC clusters require you to check out licenses for proprietary software. Also, some require requests to access some filesystems.

Run out of memory

```
slurmstepd: error: Detected 1 oom_kill event in StepId=30869.0.
Some of the step tasks have been OOM Killed.
srun: error: ku45: task 4: Out Of Memory
srun: Terminating StepId=30869.0
```

Interactive Jobs

- Most HPC jobs are run in asynchronous **batch jobs**. Submit the job and wait until it runs and completes.

- **Interactive** jobs allow you to get a live session on a set of resources for running tests, debugging, understanding resource usage, etc.

- In **Slurm**, this is achieved with ***salloc*:**

```
[27 ewalter@kuro ~ ]$salloc -N 1 -n 4 -t 3:00:00
salloc: Granted job allocation 30548
salloc: Nodes ku36 are ready for job
[1 ewalter@ku36 ~ ]$
```

- Now I am on a node with 4 cores and 3 hrs walltime.

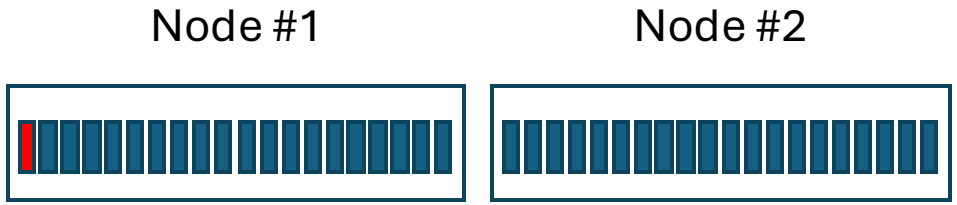
- There are two downsides to interactive jobs:
  1. You may have to wait a long time to get the job started. Adding a command line for mail to be sent when a job starts can help with this (**`--mail-type=BEGIN`**)
  2. If you lose connection to the session, the job will die.

Serial vs. Parallel Jobs

```
#!/bin/tcsh
#SBATCH --job-name=serial
#SBATCH -N 1 -n 1
#SBATCH -mem=128G
#SBATCH -t 0:30:00


./a.out_serial
```

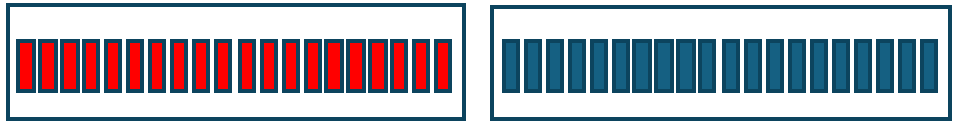
Serial job : uses one core on one node




```
#!/bin/tcsh
#SBATCH --job-name=shrdparallel
#SBATCH -N 1 -n 8
#SBATCH -mem=128G
#SBATCH -t 0:30:00

./a.out_shmparallel
```

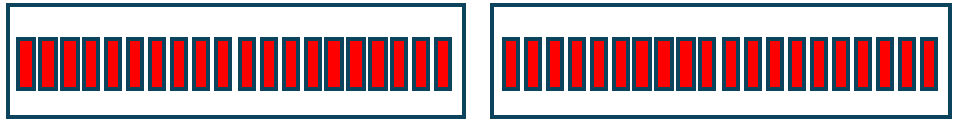
Shared-memory parallel : uses multiple cores on one node ; OpenMP



```
#!/bin/tcsh
#SBATCH --job-name=distparallel
#SBATCH -N 4 --ntasks-per-node=20
#SBATCH -t 0:30:00

srun a.out
```

Distributed-memory parallel: multiple cores on multiple nodes. MPI or MPI+OpenMP (hybrid)



More Advanced Script

```
#!/bin/bash
#SBATCH --job-name=finaltests
#SBATCH --nodes=4 --ntasks-per-node 32
#SBATCH --time=1-0


for i in `cat LIST`
do
    echo "starting run $i `date`"
    mkdir run_$i
    cd run_$i
    srun ./a.out < input_$i > OUTPUT_$i
    echo "$i run finished `date`"

    echo "run $i energy = "
    grep ENERGY OUTPUT_$i

    # gzip output file
    gzip OUTPUT_$i

    cd ..
done
```

This script uses bash syntax

Loop over list of tokens from "LIST"

Print which run I am starting with date
Make directory for run
Cd into directory
Run calculation

Print the energy of this calculation

Gzip the output file

Go back up one level

```
[33 ewalter@kuro ~ ]$cat LIST
1
3
5
6

[34 ewalter@kuro ~ ]$ls input*
input_1 input_2 input_3 input_4
input_5 input_6 input_7
```

Job Arrays in SLURM

Job arrays

```
#!/bin/tcsh
#SBATCH --job-name=multi-serial
#SBATCH -N 1 -n1
#SBATCH -mem=128G
#SBATCH -t 0:30:00
#SBATCH -a 1-4



mkdir job_$SLURM_ARRAY_TASK_ID

cd job_$SLURM_ARRAY_TASK_ID

../a.out_serial input_$SLURM_ARRAY_TASK_ID
```

Job arrays allow you define a range of values to use in a series of related jobs.

#SBATCH –a 1-4

Will submit 4 jobs with suffixes 1 through 4. In each job then the following variables are defined for you:

SLURM_ARRAY_TASK_ID - Job array ID (index) number.
SLURM_ARRAY_JOB_ID - Job array's master job ID number.
SLURM_ARRAY_TASK_COUNT - Total number of tasks in a job array.
SLURM_ARRAY_TASK_MAX - Job array's maximum ID (index) number.
SLURM_ARRAY_TASK_MIN - Job array's minimum ID (index) number.
SLURM_ARRAY_TASK_STEP - Job array's index step size.

These values can then be used to run jobs with input_1 through input_4.

USE a chat bot for scripts from scratch